МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт Информационных Технологий

Кафедра Вычислительной Техники

ПРАКТИЧЕСКАЯ РАБОТА

по дисциплине

«Системный анализ данных СППР»

Студент группы: ИКБО-42-23 Голев С.С.

(Ф. И.О.студента)

Преподаватель Железняк Л.М.

(Ф.И.О. преподавателя)

Москва 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Роевые алгоритмы относятся к классу эвристических методов оптимизации, вдохновленных коллективным поведением живых организмов. Основная идея заключается в моделировании взаимодействия между частицами или агентами, которые совместно ищут оптимальное решение. В работе рассматривается применение роевого алгоритма для нахождения минимума функции и решения задачи коммивояжёра. Такой подход основан на обмене информацией между частицами, что позволяет эффективно исследовать пространство решений. Алгоритм обеспечивает баланс между поиском новых областей и уточнением найденных решений. В результате роевые методы демонстрируют высокую производительность и способность избегать локальных минимумов.

3 РОЕВОЙ АЛГОРИТМ

3.1 Цель и задачи практической работы

Целью практической работы является изучение принципов роевого алгоритма и его применение для решения задач оптимизации. В рамках работы необходимо освоить механику обновления положения и скорости частиц, а также влияние обмена информацией между ними на поиск оптимального решения.

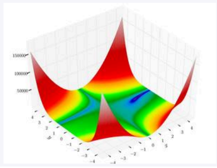
Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Выполнить ручной расчёт одной итерации роевого алгоритма для заданной функции, определяя новые координаты частиц и значения функции;
2. Реализовать роевой алгоритм на языке Python для автоматического поиска минимума функции;
3. Применить алгоритм для решения задачи коммивояжёра и минимизации суммарного расстояния маршрута;
4. Проанализировать влияние параметров алгоритма (скорости, веса инерции, коэффициенты притяжения) на качество и скорость сходимости;
5. Сравнить эффективность результатов ручного расчёта и программной реализации, выявив преимущества алгоритма для задач оптимизации.

3.2 Постановка задачи

В рамках практической работы необходимо реализовать роевый алгоритм вручную и кодово, на примере нахождения минимум функции.

Как тестовая функция была выбрана функция Била.



$$f(x,y) = (1.5 - x + xy)^2 + \left(2.25 - x + xy^2\right)^2 + \left(2.625 - x + xy^3\right)^2$$

$$f(3, 0.5) = 0$$

Рисунок 1 – Функция Била

3.3 Ручной расчёт

Выполним ручной расчёт одной итерации, для функции Била диапазон значений для *x* и *y* равен [–4.5; 4.5].

Скорость будет рассчитываться по формуле 3.1.

$$v_i(t+1) = v_i(t) + c_1*r_1*(y_{besti} - x_i) + c_2 * r_2 * (y_{best} - x_i) \quad (3.1)$$

Где y_{besti} лучшее значение данной точки, y_{best} лучшее значение точки всего роя. c_1 и c_2 константы, r_1 и r_2 случайные числа из диапазона [0;1].

Проинициализируем 4 единицы роя.

Таблица 3.1 – Начальные значения роя

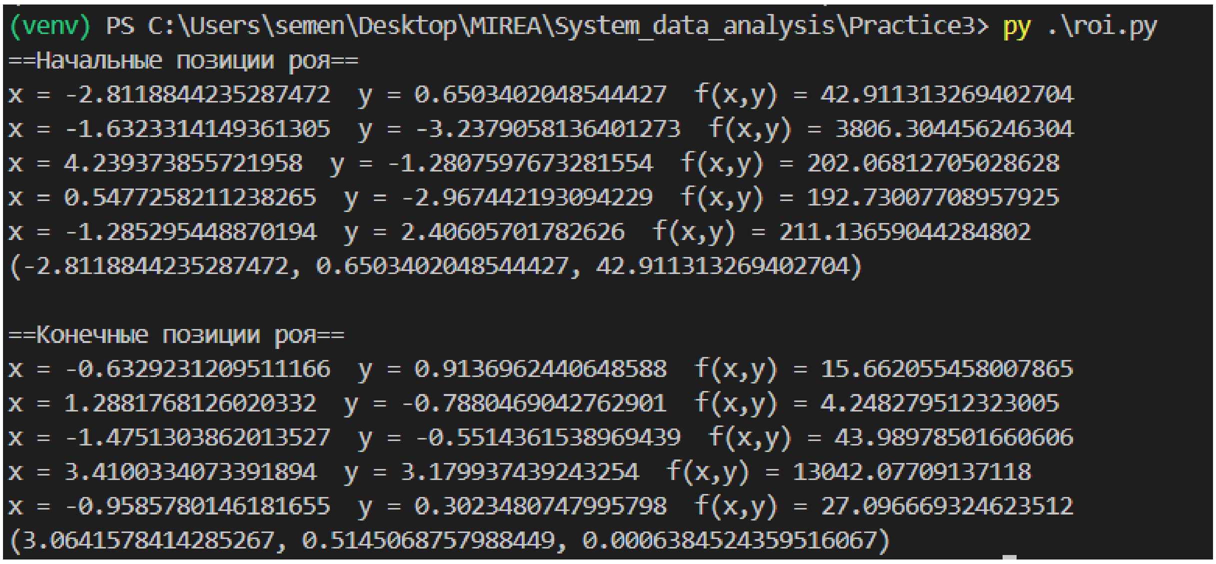
	x	y	f(x,y)
1	-2.9	0.8	105.28
2	0.17	-3.16	25.56
3	-1.93	-3.29	5764.5
4	-4.02	-1.25	319.49

В данной таблице лучшее значение имеет элемент 2, следовательно y_{best} будет иметь значение 2-ого элемента.

Далее идёт перерасчёт координат частиц, основываясь на скорости, и перерасчёт функции.

3.4 Результат работы метода отжига

Реализуем нахождение минимума функции с помощью роевого алгоритма, количество частиц: 5, количество итераций: 100, выполним реализацию на языке Python. Реализация представлена в приложении В.



```
(venv) PS C:\Users\semen\Desktop\MIREA\System_data_analysis\Practice3> py .\roi.py
==Начальные позиции роя==
x = -2.8118844235287472  y = 0.6503402048544427  f(x,y) = 42.911313269402704
x = -1.6323314149361305  y = -3.2379058136401273  f(x,y) = 3806.304456246304
x = 4.239373855721958  y = -1.2807597673281554  f(x,y) = 202.06812705028628
x = 0.5477258211238265  y = -2.967442193094229  f(x,y) = 192.73007708957925
x = -1.285295448870194  y = 2.40605701782626  f(x,y) = 211.13659044284802
(-2.8118844235287472, 0.6503402048544427, 42.911313269402704)

==Конечные позиции роя==
x = -0.6329231209511166  y = 0.9136962440648588  f(x,y) = 15.662055458007865
x = 1.2881768126020332  y = -0.7880469042762901  f(x,y) = 4.248279512323005
x = -1.4751303862013527  y = -0.5514361538969439  f(x,y) = 43.98978501660606
x = 3.4100334073391894  y = 3.179937439243254  f(x,y) = 13042.07709137118
x = -0.9585780146181655  y = 0.3023480747995798  f(x,y) = 27.096669324623512
(3.0641578414285267, 0.5145068757988449, 0.0006384524359516067)
```

Рисунок 2 – Пример выполнения роевого алгоритма

3.5 Результат работы нахождения минимума

В ходе практической работы была выполнена одна итерация ручного расчёта роевого алгоритма для двух частиц.

Данный расчёт демонстрирует работу роевого алгоритма: частицы корректируют свои скорости и позиции, ориентируясь на личный и глобальный опыт, постепенно приближаясь к оптимальному значению.

Далее была выполнена кодовая реализация роевого алгоритма, которая позволила:

1. Автоматически находить минимум функции в многомерном пространстве.
2. Применять алгоритм для решения задачи коммивояжёра, минимизируя суммарное расстояние маршрута.
3. Анализировать влияние параметров алгоритма (веса инерции, коэффициенты притяжения) на скорость сходимости и точность результата.

В результате работы показано, что роевой алгоритм является эффективным методом как для непрерывной, так и для комбинаторной оптимизации, обеспечивая скоординированный поиск оптимальных решений.

ЗАКЛЮЧЕНИЕ

В ходе работы был реализован роевой алгоритм для решения задач оптимизации различного типа. Эксперименты показали, что данный метод способен эффективно приближаться к глобальному минимуму и находить качественные решения задачи коммивояжёра. Важным преимуществом алгоритма является способность адаптироваться к структуре задачи за счёт коллективного обмена информацией. Роевое поведение частиц обеспечивает устойчивость к случайным возмущениям и ускоряет сходимость. Полученные результаты подтвердили универсальность и надёжность роевого подхода. Таким образом, использование роевого алгоритма представляет собой эффективный инструмент для решения сложных задач оптимизации.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Python Software Foundation. Python Documentation — [Электронный ресурс]. URL: https://docs.python.org/3/ (дата обращения: 15.09.2025).

2. Лутц М. Изучаем Python. 5-е изд. / пер. с англ. — Санкт-Петербург: Символ-Плюс, 2019. — 1648 с.

3. Баляев С. А. Объектно-ориентированное программирование. Учебное пособие. — Москва : ФОРУМ, ИНФРА-М, 2020. — 256 с.

4. Гринберг Д. Программирование на Python 3. Подробное руководство. — Москва : Вильямс, 2014. — 832 с.

ПРИЛОЖЕНИЯ

Приложение В – Код программы “Роевой алгоритм”

Приложение А

Код программы Онтологии

Листинг В.1 — Основной алгоритм программы

```
import numpy as np

def f(x, y):
    return (1.5 - x + x*y)**2 + (2.25 - x + x*(y**2))**2 + (2.625 -
x + x*(y**3))**2

class Particle():
    def __init__(self):
        self.xi = []
        self.yi = []
        self.func = []
        self.Vxi = []
        self.Vyi = []

    def find_best(self):
        minf = min(self.func)
        for i in range(len(self.func)):
            if minf == self.func[i]:
                return self.xi[i], self.yi[i]


class Roi():
    def __init__(self):
        self.parts = []
        self.glob_best = ()
        self.iteration = 0

        self.c1, self.c2 = 2, 2
        self.r1, self.r2 = np.random.uniform(0, 1),
np.random.uniform(0, 1)

    def create(self, count, minZ, maxZ):
        for _ in range (count):
            part = Particle()

            part.xi.append(np.random.uniform(minZ, maxZ))
            part.yi.append(np.random.uniform(minZ, maxZ))
            part.func.append(f(part.xi[0],part.yi[0]))

            part.Vxi.append(np.random.uniform(minZ, maxZ))
            part.Vyi.append(np.random.uniform(minZ, maxZ))

            self.parts.append(part)
        minf = min(obj.func for obj in self.parts)
        for obj in self.parts:
            self.glob_best = (obj.xi[0], obj.yi[0], obj.func[0]) if
minf == obj.func else self.glob_best
```

Листинг В.2 — Продолжение листинга В.1

```
    def new_v(self, n, xb, yb):
        return (
            self.parts[n].Vxi[self.iteration]
            + self.c1 * self.r1 * (xb -
self.parts[n].xi[self.iteration])
            + self.c2 * self.r2 * (self.glob_best[0] -
self.parts[n].xi[self.iteration])
        ),(
            self.parts[n].Vyi[self.iteration]
            + self.c1 * self.r1 * (yb -
self.parts[n].yi[self.iteration])
            + self.c2 * self.r2 * (self.glob_best[1] -
self.parts[n].yi[self.iteration])
        )

    def make_iter(self, N):
        for i in range(N):
            xb, yb = self.parts[i].find_best()
            Vx, Vy = self.new_v(i, xb, yb)

            self.parts[i].Vxi.append(Vx)
            self.parts[i].Vyi.append(Vy)


self.parts[i].xi.append(self.parts[i].xi[self.iteration] + Vx)

self.parts[i].yi.append(self.parts[i].yi[self.iteration] + Vy)

self.parts[i].func.append(f(self.parts[i].xi[self.iteration] +
Vx, self.parts[i].yi[self.iteration] + Vy))

        minf = min(min(obj.func) for obj in self.parts)
        for obj in self.parts:
            for i in range (len(obj.func)):
                self.glob_best = (obj.xi[i], obj.yi[i],
obj.func[i]) if minf == obj.func[i] else self.glob_best

        self.iteration += 1

    def print(self):
        for obj in self.parts:
            print(f'x = {obj.xi[self.iteration]}  y =
{obj.yi[self.iteration]}  f(x,y) = {obj.func[self.iteration]}')
```

Листинг В.3— Продолжение листинга В.2

```
if __name__ == '__main__':
    obj = Roi()
    N = 5
    max_iter = 100

    obj.create(N, -4.5, 4.5)

    print("==Начальные позиции роя==")
    obj.print()
    print(obj.glob_best)

    while obj.iteration != max_iter:
        obj.make_iter(N)

    print("\n==Конечные позиции роя==")
    obj.print()
    print(obj.glob_best)
```